#define ARRAY\_SIZE 1024 /// 1024, 4096, 8192, etc

#define BUBBLE\_SORT 1

#define SELECTION\_SORT 2

#define INSERTION\_SORT 3

#define MERGE\_SORT 4

void print\_array(int \* ptr\_array, int size);

void sort\_array(int \* ptr\_array, int size, int sort\_type);

void merge\_sort\_array(int \* ptr\_array, int size);

int main()

{

srand((unsigned int)time(NULL)); /// Random number generator seeded with value of time

/\*\* Initialize an Integer Array and initialize it with random numbers between 0-255 \*\*/

int my\_array[ARRAY\_SIZE];

for(int i=0; i<ARRAY\_SIZE; i++)

{

my\_array[i] = rand()%256;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

print\_array(my\_array, ARRAY\_SIZE);

printf("\n\n");

// Start measuring time

struct timeval begin, end;

gettimeofday(&begin, 0);

/// do the sorting part here

sort\_array(my\_array,ARRAY\_SIZE,BUBBLE\_SORT);

// Stop measuring time and calculate the elapsed time

gettimeofday(&end, 0);

print\_array(my\_array, ARRAY\_SIZE);

long seconds = end.tv\_sec - begin.tv\_sec;

long microseconds = end.tv\_usec - begin.tv\_usec;

double elapsed = seconds + microseconds\*1e-6;

printf("\n\nTime measured: %f Seconds.\n", elapsed);

printf("\nTime measured: %lu Microseconds.\n", microseconds);

return 0;

}

void print\_array(int \* ptr\_array, int size)

{

for(int i=0; i<size; i++)

{

printf("%3d\t", \*(ptr\_array+i));

}

}

void sort\_array(int \* ptr\_array, int arr\_size, int sort\_type)

{

switch(sort\_type)

{

case BUBBLE\_SORT:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Implement Bubble Sort Here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{

for(int k = arr\_size; k>0; k--)

{

for(int i=0; i<k-1; i++)

{

if(\*(ptr\_array+i)>\*(ptr\_array+i+1))

{

int temp = \*(ptr\_array+i+1);

\*(ptr\_array+i+1) = \*(ptr\_array+i);

\*(ptr\_array+i) = temp;

}

}

}}

break;

case SELECTION\_SORT:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Implement Selection Sort Here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{

int i;

for (int s = 0; s < arr\_size - 1; s++)

{

int min\_idx = s;

for ( i = s + 1; i < arr\_size; i++)

{

if (ptr\_array[i] < ptr\_array[min\_idx])

min\_idx = i;

}

if((ptr\_array[i])>(ptr\_array[min\_idx]))

{int temp = (ptr\_array[min\_idx]);

(ptr\_array[min\_idx]) = (ptr\_array[i]);

(ptr\_array[i]) = temp;

}

}}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

break;

case INSERTION\_SORT:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Implement Insertion Sort Here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{

for (int step = 1; step < arr\_size; step++)

{

int key = ptr\_array[step];

int j = step - 1;

// Compare key with each element on the left of it until an element smaller than

// it is found.

// For descending order, change key<array[j] to key>array[j].

while (key < ptr\_array[j] && j >= 0) {

ptr\_array[j + 1] = ptr\_array[j];

--j;

}

ptr\_array[j + 1] = key;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

break;

case MERGE\_SORT:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Call Merge Sort Function Here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{

if(arr\_size == 1)

return;

int size1 = arr\_size/2;

int size2;

if(arr\_size % 2 == 1)

size2 = (arr\_size/2)+1;

else

size2 = (arr\_size/2);

merge\_sort(\*ptr\_array, size1);

merge\_sort(\*ptr\_array+(arr\_size/2), size2);

merge(\*ptr\_array, size1, \*ptr\_array+(arr\_size/2), size2);

return;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

break;

}

void merge(int \* ptr\_arrA, int sizeA, int \* ptr\_arrB, int sizeB)

{

int \* ptr\_arrC = (int \*) malloc(sizeof(int)\*(sizeA+sizeB));

int indexA = 0;

int indexB = 0;

int indexC = 0;

while((indexA < sizeA)&&(indexB<sizeB))

{

if((\*(ptr\_arrA+indexA)) > (\*(ptr\_arrB+indexB)))

{ \*(ptr\_arrC + indexC) = \*(ptr\_arrB+indexB);

indexB ++;

indexC ++; }

else

{ \*(ptr\_arrC + indexC) = \*(ptr\_arrA+indexA);

indexA ++;

indexC ++; }

}

while(indexA < sizeA)

{ \*(ptr\_arrC + indexC) = \*(ptr\_arrA+indexA);

indexA ++;

indexC ++; }

while(indexB < sizeB)

{ \*(ptr\_arrC + indexC) = \*(ptr\_arrB+indexB);

indexB ++;

indexC ++; }

for(indexC = 0; indexC < (sizeA+sizeB); indexC ++)

\*(ptr\_arrA + indexC) = \*(ptr\_arrC+indexC);

free(ptr\_arrC); /// Delete the previously allocated memory

return;

}}